openest Documentation

Author

Nov 13, 2018

Contents

OpenEst is a library created by the Climate Impact Lab team.

This code is open source and available on github.

We can add any additional information about the library here. Please make suggestions.

CHAPTER 1

Intro and Setup

This is where the introduction to our libraries will go. Ideally, this will be a short overview of the library, with one or two very simple use cases. It will, hopefully, answer the question: 'Why should I use this tool?'

1.1 Installation

This part of the documentation covers the installation of OpenEst.

This library depends on numpy and scipy. In addition, to use Mean-Size hierarchical sampling, the emcee library must be installed.

`pip install numpy`

`pip install scipy`

`pip install emcee`

Install the package by calling `python setup.py install` (or use *develop* rather than install if you'll be editing the code).

CHAPTER 2

The API Documentation / Guide

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

2.1 openest package

2.1.1 Subpackages

openest.generate

Submodules

openest.generate.calculation module

class openest.generate.calculation.**Application**(*region*)
Bases: object

done()

push(*ds*)
Returns an interator of (yyyy, value, . . .).

class openest.generate.calculation.**ApplicationByChunks**(*region*)
Bases: *openest.generate.calculation.Application*

push(*ds*)
Returns an interator of (yyyy, value, . . .).

push_saved(*ds*)
Returns an interator of (yyyy, value, . . .). Removes used daily values from saved.

class openest.generate.calculation.**ApplicationByIrregular**(*region*, *func*, **args*, ***kwargs*)
Bases: *openest.generate.calculation.Application*

push (*ds*)
Returns an interator of (yyyy, value, . . .).

class openest.generate.calculation.**ApplicationByYear** (*region*, *func*, **args*, ***kwargs*)
Bases: *openest.generate.calculation.ApplicationByChunks*

push_saved (*ds*)
Returns an interator of (yyyy, value, . . .). Removes used daily values from saved.

class openest.generate.calculation.**ApplicationEach** (*region*, *func*, *finishfunc=<function <lambda>>*, **args*, ***kwargs*)
Bases: *openest.generate.calculation.Application*

Pass every set of values to the calculation for a value.

done ()

push (*ds*)
Returns an interator of (yyyy, value, . . .).

class openest.generate.calculation.**ApplicationPassCall** (*region*, *subapp*, *handler*, **handler_args*, ***handler_kw*)
Bases: *openest.generate.calculation.Application*

Apply a non-enumerator to all elements of a function. if unshift, tack on the result to the front of a sequence of results. Calls func with each year and value; returns the newly computed value

push (*ds*)
Returns an interator of (yyyy, value, . . .).

class openest.generate.calculation.**Calculation** (*unitses*)
Bases: object

apply (*region*, **args*, ***kwargs*)

cleanup ()

column_info ()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe ()
Returns dictionary containing: - input_timerate: expected time rate of data, day, month, year, or any - output_timerate: expected time rate of data, day, month, year, or same - arguments: a list of subclasses of arguments.ArgumentType, describing each constructor argument - description: text description

enable_deltamethod ()
When applied, yield will contain arrays of coefficient multiplicands as a vector the length of the CSVV coefficients.

format (*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

test ()

class openest.generate.calculation.**CustomFunctionalCalculation** (*subcalc*, *from_units*, *to_units*, *unshift*, **handler_args*, ***handler_kw*)
Bases: *openest.generate.calculation.FunctionalCalculation*, *openest.generate.calculation.Application*

Calculation that creates a copy of itself for an application.

apply (*region*, **args*, ***kwargs*)

done ()

donehandler (**allargs*, ***allkwargs*)

init_apply ()

push (*ds*)
Returns an interator of (yyyy, value, …).

pushhandler (*ds*, **allargs*, ***allkwargs*)

class openest.generate.calculation.**FunctionalCalculation** (*subcalc*, *from_units*, *to_units*, *unshift*, **handler_args*, ***handler_kw*)
Bases: *openest.generate.calculation.Calculation*

Calculation that calls a handler when it's applied.

apply (*region*, **args*, ***kwargs*)

cleanup ()

enable_deltamethod ()
When applied, yield will contain arrays of coefficient multiplicands as a vector the length of the CSVV coefficients.

format (*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

format_handler (*substr*, *lang*, **handler_args*, ***handler_kw*)

handler (*year*, *result*, **handler_args*, ***handler_kw*)

openest.generate.curvegen module

class openest.generate.curvegen.**ConstantCurveGenerator** (*indepunits*, *depenunit*, *curve*)
Bases: *openest.generate.curvegen.CurveGenerator*

format_call (*lang*, **args*)

get_curve (*region*, *year*, **args*, ***kw*)
Returns an object of type Curve.

class openest.generate.curvegen.**CurveGenerator** (*indepunits*, *depenunit*)
Bases: object

format_call (*lang*, **args*)

get_curve (*region*, *year*, **args*, ***kw*)
Returns an object of type Curve.

class openest.generate.curvegen.**DelayedCurveGenerator** (*curvegen*)
Bases: *openest.generate.curvegen.CurveGenerator*

format_call (*lang*, **args*)

get_curve (*region*, *year*, **args*, ***kwargs*)
Returns an object of type Curve.

get_next_curve (*region*, *year*, **args*, ***kwargs*)

class openest.generate.curvegen.**TransformCurveGenerator**(*transform*, *description*, **curvegens*)

Bases: `openest.generate.curvegen.CurveGenerator`

format_call(*lang*, **args*)

get_curve(*region*, *year*, **args*, ***kw*)
Returns an object of type Curve.

get_lincom_terms(*region*, *year*, *predictors*)

get_lincom_terms_simple(*predictors*, *covariates={}*)

openest.generate.daily module

class openest.generate.daily.**ApplyCurve**(*curvegen*, *unitses*, *names*, *titles*, *descriptions*)
Bases: `openest.generate.calculation.Calculation`

apply(*region*, **args*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

class openest.generate.daily.**AverageByMonth**(*model*, *units*, *func=<function <lambda>>*, *pval=0.5*)
Bases: `openest.generate.calculation.Calculation`

apply(*region*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.daily.**MonthlyDayBins**(*model*, *units*, *pval=0.5*, *weather_change=<function <lambda>>*)
Bases: `openest.generate.calculation.Calculation`

apply(*region*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.daily.**PercentWithin**(*endpoints*)
Bases: `openest.generate.calculation.Calculation`

apply(*region*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

class openest.generate.daily.**YearlyAverageDay**(*units*, *curvegen*, *curve_description*, *weather_change=<function <lambda>>*, *norecord=False*)

Bases: `openest.generate.calculation.Calculation`

apply(*region*, **args*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*)

Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.daily.**YearlyDayBins**(*model*, *units*, *pval=0.5*)

Bases: `openest.generate.calculation.Calculation`

apply(*region*, **args*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*)

Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.daily.**YearlyDividedPolynomialAverageDay**(*units*, *curvegen*, *curve_description*, *weather_change=<function <lambda>>*)

Bases: `openest.generate.calculation.Calculation`

apply(*region*, **args*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

class openest.generate.daily.**YearlySumDay**(*units*, *curvegen*, *curve_description*, *weather_change=<function <lambda>>*, *norecord=False*)

Bases: `openest.generate.daily.YearlyAverageDay`

apply(*region*, **args*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*)

Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

openest.generate.functions module

class openest.generate.functions.**AuxillaryResult**(*subcalc_main*, *subcalc_aux*, *aux-name*)

Bases: `openest.generate.calculation.Calculation`

Produce an additional output, but then pass the main result on.

apply (*region*, **args*, ***kwargs*)

column_info ()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe ()

format (*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.functions.**AuxillaryResultApplication** (*region*, *subapp_main*, *subapp_aux*)
Bases: *openest.generate.calculation.Application*

Perform both main and auxillary calculation, and order as main[0], aux, main[1:]

done ()

push (*ds*)
Returns an interator of (yyyy, value, ...).

class openest.generate.functions.**ConstantScale** (*subcalc*, *coeff*)
Bases: *openest.generate.calculation.Calculation*

apply (*region*, **args*, ***kwargs*)

column_info ()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe ()

format (*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.functions.**Exponentiate** (*subcalc*)
Bases: *openest.generate.calculation.Calculation*

apply (*region*, **args*, ***kwargs*)

column_info ()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe ()

format (*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.functions.**InstaZScore** (*subcalc*, *lastyear*, *units='z-score'*)
Bases: *openest.generate.calculation.CustomFunctionalCalculation*

Collects up to *baseyear* of values and then uses them to represent all values as a z-score.

column_info ()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe ()

init_apply ()

pushhandler (*ds*, *lastyear*)
Returns an interator of (yyyy, value, ...).

class openest.generate.functions.**Instabase**(*subcalc*, *baseyear*, *func=<function <lambda>>*, *units='portion'*, *skip_on_missing=True*)

Bases: openest.generate.calculation.CustomFunctionalCalculation

Re-base the results of make_generator(…) to the values in baseyear baseyear is the year to use as the 'denominator'; None for the first year Default func constructs a porportional change; x - y makes simple difference. skip_on_missing: If we never encounter the year and this is false,

still print out the existing results.

Tacks on the value to the front of the results

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

donehandler(*baseyear*, *func*, *skip_on_missing*)

format_handler(*equation*, *lang*, *baseyear*, *func*, *skip_on_missing*)

init_apply()

pushhandler(*ds*, *baseyear*, *func*, *skip_on_missing*)

Returns an interator of (yyyy, value, …).

class openest.generate.functions.**Positive**(*subcalc*)

Bases: openest.generate.calculation.Calculation

Return 0 if subcalc is less than 0

apply(*region*, **args*, ***kwargs*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

class openest.generate.functions.**Scale**(*subcalc*, *scale_dict*, *from_units*, *to_units*, *func=<function <lambda>>*, *latexpair=('\bar{I}', 'Region-specific scaling')*)

Bases: openest.generate.calculation.Calculation

apply(*region*, **args*, ***kwargs*)

column_info()

Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*, **args*, ***kwargs*)

Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.functions.**SpanInstabase**(*subcalc*, *year1*, *year2*, *func=<function <lambda>>*, *units='portion'*, *skip_on_missing=True*)

Bases: openest.generate.functions.Instabase

Re-base the results of a calculation to the average of values between two years. Default func constructs a porportional change; x - y makes simple difference. skip_on_missing: If we never encounter the year and this is false,

still print out the existing results.

static describe()

format_handler(*equation*, *lang*, *baseyear*, *func*, *skip_on_missing*)

init_apply()

pushhandler(*ds*, *baseyear*, *func*, *skip_on_missing*)
Returns an interator of (yyyy, value, . . .).

class openest.generate.functions.**Sum**(*subcalcs*)
Bases: *openest.generate.calculation.Calculation*

apply(*region*, **args*, ***kwargs*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

enable_deltamethod()
When applied, yield will contain arrays of coefficient multiplicands as a vector the length of the CSVV coefficients.

format(*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

class openest.generate.functions.**Transform**(*subcalc*, *from_units*, *to_units*, *func*, *description*, *long_description*)
Bases: *openest.generate.calculation.Calculation*

apply(*region*, **args*, ***kwargs*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static describe()

format(*lang*, **args*, ***kwargs*)
Returns a dictionary of FormatElements. Only keys in the tree of dependencies will be output.

openest.generate.latextools module

openest.generate.latextools.**call**(*func*, *description=None*, **args*)
Return a representation of this call. Any elements in args can be given their own FormatElements in the final dictionary.

openest.generate.latextools.**english_function**(*func*, **args*)

openest.generate.latextools.**latex_function**(*func*, **args*)

openest.generate.retrieve module

openest.generate.retrieve.**any_from_url**(*url*)
Returns a model retrieved from the argument *url*

any_from_url is a wrapper around *from_url()*. It returns a model chosen by *choose_model()*. Therefore, the file reader returned by *from_url()* must have one of the allowed model types as the first four characters in the document.

Parameters **url** (*str*) – URL of file to retrieve

Returns Model chosen by *choose_model()*

Return type object

`openest.generate.retrieve.`**`choose_model`**`(`*fp*, *source=None*`)`

Reads a file object and returns a model based on file header

The file is converted into a *`BinModel`*, *`DDPModel`*, or *`SplineModel`* depending on the first four characters of the file.

To use choose_model, the first four characters of the file reader object must be one of the following:

- `bin1`, in which case a *`BinModel`* will be returned,
- `ddp1` or `ddp2`, in which case a *`DDPModel`* will be returned, or
- `spp1`, in which case a *`SplineModel`* will be returned.

If the model type is not one of the types listed above, a *BaseException* will be raised.

Todo:

- **Change exception type - sublcassing `BaseException` is not PEP compliant.** Custom exceptions should inherrit from `Exception` or other built-in exceptions. This is so that `except Exception` will catch all exceptions except for `KeyboardInterrupt` and `SystemExit`, which are not errors, but user-triggered events. See PEP-352.

Parameters

- **`fp`** (*`file reader object`*) – file reader object to be converted into a model.
- **`source`** (*`str`*) – Meta-information about url the file was recovered from

Returns Model of class *`BinModel`*, *`DDPModel`*, or *`SplineModel`*.

Return type object

`openest.generate.retrieve.`**`ddp_from_url`**`(`*url*`)`

Returns a *`DDPModel`* from the argument *url*

`openest.generate.retrieve.`**`from_url`**`(`*url*, *create_func*`)`

Returns a `StringIO.StringIO` buffer with the contents of the response from *url*

Todo:

- response from *urllib2.urlopen(req)* is already a buffer. Is writing to a new buffer necessary?

`openest.generate.retrieve.`**`spline_from_url`**`(`*url*`)`

Returns a *`SplineModel`* from the argument *url*

openest.generate.shortterm module

`class` `openest.generate.shortterm.`**`InstaZScoreApply`**`(`*units*, *curve*, *curve_description*, *lasttime*, *weather_change=<function <lambda>>*`)`

Bases: *`openest.generate.calculation.Calculation`*, *`openest.generate.calculation.Application`*

`apply``(`*region*, **args*, ***kwargs*`)`

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static **describe**()

push(*time*, *weather*)
Returns an interator of (yyyy, value, ...).

class openest.generate.shortterm.**MonthlyClimateApply**(*units*, *curve*, *curve_description*, *monthmeans*, *regions*, *weather_change=<function <lambda>>*)
Bases: *openest.generate.calculation.Calculation*

apply(*region*, **args*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static **describe**()

class openest.generate.shortterm.**MonthlyZScoreApply**(*units*, *curve*, *curve_description*, *monthmeans*, *monthsdevs*, *regions*, *weather_change=<function <lambda>>*)
Bases: *openest.generate.calculation.Calculation*, *openest.generate.calculation.Application*

apply(*region*, **args*, ***kwargs*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static **describe**()

push(*time*, *weather*)
Returns an interator of (yyyy, value, ...).

class openest.generate.shortterm.**SingleWeatherApply**(*units*, *curve*, *curve_description*, *weather_change=<function <lambda>>*)
Bases: *openest.generate.calculation.Calculation*

apply(*region*, **args*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static **describe**()

class openest.generate.shortterm.**SplitByMonth**(*subcalc*)
Bases: *openest.generate.calculation.Calculation*

apply(*region*, **args*, ***kwargs*)

column_info()
Returns an array of dictionaries, with 'name', 'title', and 'description'.

static **describe**()

openest.generate.stdlib module

openest.generate.weathertools module

openest.generate.weathertools.**combo_effects**(*effect_dicts*, *scale_gens*)

openest.generate.weathertools.**date_to_datestr**(*date*)

openest.generate.weathertools.**get_crop_calendar**(*cropfile*)

openest.generate.weathertools.**growing_seasons_daily_ncdf**(*yyyyddd*, *weather*, *plantday*, *harvestday*)

openest.generate.weathertools.**growing_seasons_mean_ncdf**(*yyyyddd*, *weather*, *plantday*, *harvestday*)

openest.generate.weathertools.**growing_seasons_mean_reader**(*reader*, *plantday*, *harvestday*)

openest.generate.weathertools.**read_scale_file**(*filepath*, *factor*)

openest.generate.weathertools.**xmap_apply_model**(*xmap*, *model*, *pval*)

openest.generate.weathertools.**yearly_daily_ncdf**(*yyyyddd*, *weather*)

openest.lincombo package

Submodules

openest.lincombo.continuous_sampled module

class openest.lincombo.continuous_sampled.**ContinuousSampled**(*func*)

Bases: `scipy.stats._distn_infrastructure.rv_continuous`

guess_ranges(*mini*, *maxi*, *count=10000*)

guess_ranges_gridded(*mini*, *maxi*, *count=10000*)

pdf(*xxs*)

Probability density function at x of the given RV.

Parameters

- **x** (*array_like*) – quantiles
- **arg2, arg3,..** (*arg1,*) – The shape parameter(s) for the distribution (see docstring of the instance object for more information)
- **loc** (*array_like, optional*) – location parameter (default=0)
- **scale** (*array_like, optional*) – scale parameter (default=1)

Returns **pdf** – Probability density function evaluated at x

Return type ndarray

prepare_draws(*mini*, *maxi*, *count=10000*)

prepare_draws_gridded(*mini*, *maxi*, *count=10000*)

rvs(*size=1*, *random_state=None*)

Random variates of given type.

Parameters

- **arg2, arg3, ..** (*arg1,*) – The shape parameter(s) for the distribution (see docstring of the instance object for more information).
- **loc** (*array_like, optional*) – Location parameter (default=0).
- **scale** (*array_like, optional*) – Scale parameter (default=1).
- **size** (*int or tuple of ints, optional*) – Defining number of random variates (default is 1).
- **random_state** (None or int or `np.random.RandomState` instance, optional) – If int or RandomState, use it for drawing the random variates. If None, rely on `self.random_state`. Default is None.

Returns **rvs** – Random variates of given *size*.

Return type ndarray or scalar

openest.lincombo.helpers module

Helper functions

`openest.lincombo.helpers.`**`check_arguments`**(*betas*, *stderrs*, *portions*)
Ensure that the parameters have the right dimensions for calculation.

`openest.lincombo.helpers.`**`issparse`**(*portions*)
Check if an array is sparse.

openest.lincombo.hiernorm module

`openest.lincombo.hiernorm.`**`alpha_given_taus`**(*betas*, *stdvars*, *portions*, *obstaus*)

`openest.lincombo.hiernorm.`**`betahat_given_taus`**(*betas*, *stdvars*, *portions*, *obstaus*)

`openest.lincombo.hiernorm.`**`get_sampled_column`**(*allvals*, *col*)

`openest.lincombo.hiernorm.`**`lincombo_hiernorm_taubyalpha`**(*betas*, *stderrs*, *portions*, *maxtau=None*, *guess_range=False*, *draws=100*)

`openest.lincombo.hiernorm.`**`lincombo_hiernorm_taubybeta`**(*betas*, *stderrs*, *portions*, *maxtaus=None*, *guess_range=False*, *draws=100*)

`openest.lincombo.hiernorm.`**`probability_tau`**(*alphas*, *taus*, *obstaus*, *betas*, *stdvars*, *portions*, *probability_prior_taus*)

`openest.lincombo.hiernorm.`**`sample_posterior`**(*betas*, *stderrs*, *portions*, *taudist*, *taus2obstaus*, *draws=100*)

openest.lincombo.hierregress module

`openest.lincombo.hierregress.`**`betahat_given_tau`**(*yy*, *stdvars*, *XX*, *tau*)

`openest.lincombo.hierregress.`**`get_sampled_column`**(*allvals*, *col*) openest.lincombo.hierregress.**lincombo_hierregress**(*yy*, *stderrs*, *XX*, *maxtau=None*, *guess_range=False*, *draws=100*)

openest.lincombo.hierregress.**lincombo_hierregress_taubybeta**(*yy*, *stderrs*, *XX*, *maxtau=None*, *guess_range=False*, *draws=100*)

openest.lincombo.hierregress.**lincombo_hierregress_taubymu**(*yy*, *stderrs*, *XX*, *maxtau=None*, *guess_range=False*, *draws=100*)

openest.lincombo.hierregress.**mu_given_tau**(*yy*, *stdvars*, *XX*, *tau*)

openest.lincombo.hierregress.**probability_tau**(*mus*, *tau*, *yy*, *stdvars*, *XX*, *probability_prior_tau*)

openest.lincombo.hierregress.**sample_posterior**(*yy*, *stderrs*, *XX*, *taudist*, *draws=100*)

openest.lincombo.montecarlo module

openest.lincombo.montecarlo.**regress_distribution**(*means*, *serrs*, *XX*, *count=1000*)

openest.lincombo.montecarlo.**regress_draws**(*means*, *serrs*, *XX*, *count=1000*)

openest.lincombo.montecarlo.**regress_summary**(*means*, *serrs*, *XX*, *count=1000*)

openest.lincombo.multi_delta module

class openest.lincombo.multi_delta.**MultivariateDelta**(*vals*)

Bases: scipy.stats._multivariate.multi_rv_frozen

pdf(*xxs*)

rvs(*size=1*, *random_state=None*)

vals()

openest.lincombo.multi_draws module

class openest.lincombo.multi_draws.**MultivariateDraws**(*draws*)

Bases: scipy.stats._multivariate.multi_rv_frozen

mean()

rvs(*size=1*, *random_state=None*)

std()

openest.lincombo.multi_normal module

class openest.lincombo.multi_normal.**MultivariateNormal**(*means*, *big_sigma*)

Bases: scipy.stats._multivariate.multi_rv_frozen

logpdf(*xxs*)

pdf(*xxs*)

rvs (*size=1*)

openest.lincombo.multi_sampled module

class openest.lincombo.multi_sampled.**MultivariateSampled** (*func*, *dims*)
Bases: scipy.stats._multivariate.multi_rv_frozen

guess_ranges (*mins*, *maxs*, *count=10000*)

guess_ranges_gridded (*mins*, *maxs*, *count=10000*)

pdf (*xxs*)

prepare_draws (*mins*, *maxs*, *count=10000*)

prepare_draws_gridded (*mins*, *maxs*, *lens*)

rvs (*size=1*, *random_state=None*)

openest.lincombo.multi_uniform module

class openest.lincombo.multi_uniform.**MultivariateUniform** (*mins*, *maxs*)
Bases: scipy.stats._multivariate.multi_rv_frozen

maxs ()

mins ()

pdf (*xxs*)

rvs (*size=1*, *random_state=None*)

openest.lincombo.pooling module

Create a pooled estimates.

The main function is *lincombo_pooled*.

openest.lincombo.pooling.**estimated_maxlintaus** (*betas*, *stderrs*, *portions*)
For use with hiernorm by-beta.

openest.lincombo.pooling.**estimated_maxtau** (*betas*, *stderrs*, *portions*)
For use with hiernorm by-alpha.

openest.lincombo.pooling.**lincombo_pooled** (*betas*, *stderrs*, *portions*)

openest.lincombo.pooling.**sum_multiply** (*sparsecol*, *densevec*)

openest.lincombo.pooling.**sum_multiply2** (*sparse*, *col1*, *col2*, *densevec*)

openest.models package

Submodules

openest.models.bin_model module

class openest.models.bin_model.**BinModel** (*xx=None*, *model=None*)
Bases: *openest.models.univariate_model.UnivariateModel*, *openest.models.*

memoizable.MemoizableUnivariate

Bin Model

A bin model represents bins of different spans, where the distribution is constant over each bin. It is a combination of information describing the bins and an underlying categorical model of one of the other types.

The underlying model is always categorical, with categories starting at 1. 0 is reserved for a future version that allows an out-of-sample distribution

The format is:

```
bin1
<x0>,<x1>,<x2>, ...
<underlying model>
```

Parameters

- **xx** (*list-like*) – List-like array of bin edges. *len(xx)* should be one more than the number of bins.
- **model** (*object*) – Statistical model used in each bin

cdf (*x*, *y*)

static combine (*one*, *two*)
Both models are BinModels

static consistent_bins (*models*)
All models are BinModels

copy ()
copy data and return BinModel with the same data

draw_sample (*x=None*)
Produce a sample value of y from the conditional distribution.

eval_pval (*x*, *p*, *threshold=0.001*)
Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

eval_pval_index (*ii*, *p*, *threshold=0.001*)

filter_x (*xx*)
Returns new *BinModel*

get_bin_at (*x*)
Returns bin containing value *x*

Parameters **x** (*numeric*) – Value to search for in binned axis

Returns Returns index of bin containing *x*. If bin is not contained in the bin range, returns `-1`.

Return type int

get_edges ()
Returns bin edges (duplicate of *get_xx()*)

get_mean (*x=None*, *index=None*)
E[Y | X]

get_sdev (*x=None*, *index=None*)
sqrt Var[Y | X]

get_xx()
returns x axis index

get_xx_centers()
returns x axis index

init_from_bin_file(*file*, *delimiter*, *status_callback=None*, *init_submodel=<function <lambda>>*)

interpolate_x(*newxx*)
Returns a copy of the model. *Does not interpolate.*

kind()
returns model type ("bin_model")

static merge(*models*)
All models are BinModels

scale_p(*a*)
Scales p-values of underlying bin models (in log_p format)

Interface to *self.model.scale_p*.

scale_y(*a*)
Scales y-axes of underlying bin models

Interface to *self.model.scale_y(a)*

to_ddp(*ys=None*)

to_points_at(*x*, *ys*)
Conditional Probability Density Evaluation

Returns unscaled probability density values for given values of x and y: $f(y | x)$.

write(*file*, *delimiter*)
Write model as delimited document to file-like object

Prepends model type (`bin1`) and bin borders (`xx`) to document written by `self.model.write`.

Parameters

- **file** (*object*) – file-like object
- **delimiter** (*str*) – Delimiter to use in file (e.g. ' ', ',')

write_file(*filename*, *delimiter*)
Write model as delimited document to filepath

Wrapper around `write()` method.

Parameters

- **filename** (*str*) – Path to file to be written
- **delimiter** (*str*) – Delimiter to use in file (e.g. ' ', ',')

openest.models.curve module

class openest.models.curve.**ClippedCurve**(*curve*, *cliplow=True*)
Bases: `openest.models.curve.UnivariateCurve`

class openest.models.curve.**CoefficientsCurve**(*coeffs*, *curve*, *xtrans=None*)
Bases: *openest.models.curve.UnivariateCurve*

A curve represented by the sum of multiple predictors, each multiplied by a coefficient.

class openest.models.curve.**CubicSplineCurve**(*knots*, *coeffs*)
Bases: *openest.models.curve.UnivariateCurve*

get_terms(*x*)
Get the set of knots-1 terms representing temperature x.

class openest.models.curve.**CurveCurve**(*xx*, *curve*)
Bases: *openest.models.curve.UnivariateCurve*

static make_linear_spline_curve(*xx*, *yy*, *limits*)

class openest.models.curve.**FlatCurve**(*yy*)
Bases: *openest.models.curve.CurveCurve*

class openest.models.curve.**LinearCurve**(*yy*)
Bases: *openest.models.curve.CurveCurve*

class openest.models.curve.**MinimumCurve**(*curve1*, *curve2*)
Bases: *openest.models.curve.UnivariateCurve*

class openest.models.curve.**OtherClippedCurve**(*clipping_curve*, *value_curve*, *clipy=0*)
Bases: *openest.models.curve.ClippedCurve*

class openest.models.curve.**PiecewiseCurve**(*curves*, *knots*, *xtrans=<function <lambda>>*)
Bases: *openest.models.curve.UnivariateCurve*

class openest.models.curve.**ProductCurve**(*curve1*, *curve2*)
Bases: *openest.models.curve.UnivariateCurve*

class openest.models.curve.**SelectiveInputCurve**(*curve*, *indices*)
Bases: *openest.models.curve.UnivariateCurve*

Assumes input is a matrix, and only pass selected input columns to child curve.

class openest.models.curve.**ShiftedCurve**(*curve*, *offset*)
Bases: *openest.models.curve.UnivariateCurve*

class openest.models.curve.**StepCurve**(*xxlimits*, *yy*, *xtrans=None*)
Bases: *openest.models.curve.CurveCurve*

class openest.models.curve.**UnivariateCurve**(*xx*)
Bases: *openest.models.univariate_model.UnivariateModel*

eval_pval(*x*, *p*, *threshold=0.001*)
Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

eval_pvals(*x*, *p*, *threshold=0.001*)

get_xx()
Listing conditional values

Provide a list of all sampled conditional values.

class openest.models.curve.**ZeroInterceptPolynomialCurve**(*xx*, *ccs*)
Bases: *openest.models.curve.UnivariateCurve*

openest.models.curve.**pos**(*x*)

openest.models.ddp_model module

class openest.models.ddp_model.**DDPModel**(*p_format=None*, *source=None*, *xx_is_categorical=False*, *xx=None*, *yy_is_categorical=False*, *yy=None*, *pp=None*, *unaccounted=None*, *scaled=True*)

Bases: *openest.models.univariate_model.UnivariateModel*, *openest.models.memoizable.MemoizableUnivariate*

Discrete-Discrete-Probability (DDP) Format

A DDP file describes a dose-response relationship with a limited collection of response outcomes. The dose and response values may be either categorical or sampled at a collection of numerical levels.

`<y-value-1>, ..., <y-value-N>` and `<x-value-1>, ..., <x-value-N>` are either strings (for named categories) or numerical values.

The format of a DDP file is:

```
<format>,<y-value-1>,<y-value-2>,...
<x-value-1>,p(y1|x1),p(y2|x1),...
<x-value-2>,p(y1|x2),p(y2|x2),...
```

Below is a sample categorical DDP file:

```
ddp1,live,dead
control,.5,.5
treated,.9,.1
```

Below is a sample numerical DDP file:

```
ddp1,-10.0,-.33333333333,3.33333333333,10.0
0.0,0.5,0.5,0.0,0.0
13.3333333333,0.0,0.5,0.5,0.0
26.6666666667,0.0,0.0,0.5,0.5
40.0,0.0,0.0,0.0,0.5
```

Parameters

- **p_format** (*str*) – Probability format. May be one of the following values:
  - ddp1 - the p(.) values are simple probabilities (0 < p(.) < 1 and sum p(y|x) = 1)
  - ddp2 - the p(.) values are log probabilities
- **source** (*str*) – Metadata attribute. Name of file this object was read in from.
- **xx_is_categorical** (*bool*) – Indicates whether `xx` is categorical. False indicates numeric data.
- **xx** (*list-like*) – X axis index
- **yy_is_categorical** (*bool*) – Indicates whether `yy` is categorical. False indicates numeric data.
- **yy** (*list-like*) – Y axis index
- **pp** (*array-like*) – underlying numpy(?) data array
- **unaccounted** (*numpy.array*) – column of remaining probability. `unaccounted = 1-sum(pp, axis=1)`.

- **scaled** (*bool*) – Indicates whether data has been scaled. If scaled, re-scale so `pp.sum(axis=1)==1`.

add_to_y(*a*)
add value a to each element of index y (numeric only)

static combine(*one*, *two*)

copy()
copy data and return DDPModel with the same data

static create_lin(*yy*, *xxs*)
Create a DDP model by supplying y index and dictionary of p-values

Parameters

- **yy** (*list-like*) – y-index labels
- **xxs** (*dict*) – dictionary keyed with x-index values with p-values for vals

draw_sample(*x=None*)
Randomly sample label from y-index using p values in row x

If x is None (default), use first row. Uses self.get_closest(x) to find matching nearest match for x-index label x

eval_pval(*x*, *p*, *threshold=0.001*)
Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

eval_pval_index(*ii*, *p*, *threshold=0.001*)

filter_x(*xx*)
Slice DDPModel data such that the values of the x index == xx

static from_file(*filename*, *delimiter*)
read DDP file from file path

get_closest(*x=None*)
return closest index on x axis

If x index is categorical, coerce x to string and find first matching index. If numeric, find the closest value.

If x is None (default), return 0

get_mean(*x=None*)
Returns the mean of the y-index labels weighted by p values in row x

If x is None (default), use first row. Uses self.get_closest(x) to find matching nearest match for x-index label x

get_sdev(*x=None*)
Returns the std dev of the y-index labels weighted by p values in row x

If x is None (default), use first row. Uses self.get_closest(x) to find matching nearest match for x-index label x

get_xx()
returns x axis index

get_yy()
returns x axis index

init_from (*file*, *delimiter*, *status_callback=None*, *source=None*)
Read DDP data set from file

init_from_other (*ddp*)
copy attributes of other DDP dataset to this one

interpolate_x (*newxx*, *kind='quadratic'*)
custom interpolation method. wrapper around scipy.interp1d.

Parameters

- **newxx** (`list-like`) – new x axis
- **kind** (`str`) – interpolation method, passed to scipy.interp1d

interpolate_y (*newyy*, *kind='quadratic'*)
custom interpolation method. wrapper around scipy.interp1d.

Parameters

- **newyy** (`list-like`) – new y axis
- **kind** (`str`) – interpolation method, passed to scipy.interp1d

kind ()
returns model type ("ddp_model")

lin_p ()
convert any DDPModel to ddp1 (linear probability) format

log_p ()
convert any DDPModel to ddp2 (log probability) format

static merge (*models*)

recategorize_x (*oldxx*, *newxx*)

rescale (*as_ddp=True*)
Can rescale non-ddp (that is, as sampling of continuous distribution)

scale_p (*a*)
coerce to ddp2 (log probability) format and scale by a

scale_y (*a*)
multiply index y (numeric only) by scale factor a

to_ddp (*ys=None*)
coerce to DDP, interpolating along y axis if necessary

transpose ()
transpose data structure

write (*file*, *delimiter*)
write CSV to file object

write_file (*filename*, *delimiter*)
write CSV to file path

openest.models.delta_model module

class openest.models.delta_model.**DeltaModel** (*xx_is_categorical=False*, *xx=None*, *locations=None*, *scale=1*)
Bases: `openest.models.univariate_model.UnivariateModel`

cdf (*xx*, *yy*)

static combine (*one*, *two*)

copy ()

draw_sample (*x=None*)
Produce a sample value of y from the conditional distribution.

filter_x (*xx*)

init_from_delta_file (*file*, *delimiter*, *status_callback=None*)

interpolate_x (*newxx*, *kind='quadratic'*)

kind ()

static merge (*models*)

scale_p (*a*)
Raising a delta function to a power makes no difference.

scale_y (*a*)
Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(rac{y}{a} | x)$.

to_points_at (*x*, *ys*)
Conditional Probability Density Evaluation

Returns unscaled probability density values for given values of x and y: $f(y | x)$.

write (*file*, *delimiter*)

write_file (*filename*, *delimiter*)

static zero_delta (*model*)

openest.models.distribution_model module

class openest.models.distribution_model.**DistributionModel** (*p_format=None*, *source=None*, *xx_is_categorical=False*, *xx=None*, *yy_is_categorical=False*, *yy=None*, *pp=None*, *unaccounted=None*, *scaled=True*)

Bases: `openest.models.ddp_model.DDPModel`

apply_as_distribution (*model*)

openest.models.features_interpreter module

Probability Features File

The probability features file has the following format:

```
dpc1,<p-header-1>,<p-header-2>,...
<x-value-1>,g_1(y | x_1),g_2(y | x_1),...
<x-value-2>,g_1(y | x_2),g_2(y | x_2),...
...
```

`<p-header>` headers can be any of the following, with the corresponding values in their rows (`<p-value-ij>`).

- `mean`: E y|x_i
- `var`: E (y|x_i - E y|x_i)^2
- `sdev`: sqrt{E (y|x_i - E y|x_i)^2}
- `skew`: E ((y|x_i - E y|x_i) / sqrt{E (y|x_i - E y|x_i)^2})^3
- `mode`: max f(y | x_i)
- numeric (0 - 1): F^{-1}(p_j|x_i)

The row headers (`<x-value>`) can be numeric, in which case a continuous spline bridges them, or categorical strings.

Below is a sample features file:

```
dpc1,mean,var
treated,0,1
control,4,4
```

class openest.models.features_interpreter.**FeaturesInterpreter**

static best_knot(*knots*, *newknots*)
Find the knot furthest from existing knots

static best_spline(*header*, *row*, *limits*)

static evaluate_spline(*header*, *row*, *spline*, *limits*)

static features_to_exponential(*header*, *row*, *limits*)

static features_to_gaussian(*header*, *row*, *limits*)

static features_to_uniform(*header*, *row*, *limits*)

static init_from_feature_file(*spline*, *file*, *delimiter*, *limits*, *status_callback=None*)

static make_conditional(*header*, *row*, *limits*)

static make_conditional_respecting(*header*, *row*, *limits*)

static skew_gaussian_construct(*ys*, *lps*, *low_segment*, *high_segment*)

static skew_gaussian_evaluate(*ys*, *lps*, *low_segment*, *high_segment*, *mean*, *lowp*, *highp*)

openest.models.generate module

openest.models.generate.**polynomial**(*lowbound*, *highbound*, *betas*, *covas*, *num=40*)

openest.models.generate.**uniform_constant**(*xx*, *yy*, *min*, *max*)

openest.models.generate.**uniform_doseless**(*start*, *end*, *height=None*)

openest.models.hierarchical_normal module

openest.models.hierarchical_normal.**draw_from_counts**(*x_counts*, *x_range*, *pval=None*)

openest.models.hierarchical_normal.**generate_thetas**(*mu_counts*, *mu_range*, *tau_counts*, *tau_range*, *count*)

openest.models.hierarchical_normal.**get_random**(*taus*, *F_tau*, *count*)

openest.models.hierarchical_normal.**helper_params**(*means*, *varis*, *tau*)

openest.models.hierarchical_normal.**p_tau_given_y**(*tau*, *means*, *varis*)

openest.models.hierarchical_normal.**simulate_normal_model**(*means*, *serrs*, *count*, *taus=None*, *do_thetas=False*)

openest.models.integral_model module

Integral model

The integral over x of another model.

class openest.models.integral_model.**IntegralModel**(*model=None*)

Bases: *openest.models.univariate_model.UnivariateModel*

copy()

eval_pval(*x*, *p*, *threshold=0.001*)

Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

get_xx()

Listing conditional values

Provide a list of all sampled conditional values.

interpolate_x(*newxx*)

kind()

scale_p(*a*)

Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

scale_y(*a*)

Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(rac{y}{a} | x)$.

write(*file*, *delimiter*)

write_file(*filename*, *delimiter*)

openest.models.mean_size_model module

Mean-Size Model

In Mean-Size models, each point is characterized only by a value and the population size that went into estimating that value. As such, it does not have enough information to generate a full distribution. It can be safely combined with other mean-size models, or approximated with a Gaussian (with a variance which is equal to the absolute value of the mean for size = 1, and a variance that decreases with the square root of the size, according to the Central Limit Theorem).

The format is:

```
msx1,mean,size
<x0>,<mean0>,<size0>
<x1>,<mean1>,<size1>
...
```

class openest.models.mean_size_model.**MeanSizeModel**(*xx_is_categorical=False*, *xx=None*, *means=None*, *sizes=None*)

Bases: *openest.models.univariate_model.UnivariateModel*

attribute_list()

static combine(*one*, *two*)

copy()

filter_x(*xx*)

get_attribute(*title*)

get_mean(*x=None*)
E[Y | X]

get_sdev(*x=None*)
sqrt Var[Y | X]

init_from_mean_size_file(*file*, *delimiter*, *status_callback=None*)

interpolate_x(*newxx*, *kind='quadratic'*)

kind()

static merge(*models*, *treatment='default'*)

scale_p(*a*)
Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

scale_y(*a*)
Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(rac{y}{a} | x)$.

write(*file*, *delimiter*)

write_file(*filename*, *delimiter*)

openest.models.memoizable module

class openest.models.memoizable.**MemoizableUnivariate**

Bases: object

eval_pval_index(*ii*, *p*, *threshold=0.001*)

get_edges()

class openest.models.memoizable.**MemoizedUnivariate**(*model*)

Bases: *openest.models.univariate_model.UnivariateModel*

copy()

eval_pval(*x*, *p*, *threshold=0.001*)

Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

eval_pvals(*xs*, *p*, *threshold=0.001*)

get_eval_pval_spline(*p*, *limits*, *threshold=0.001*, *linextrap=False*)

get_index(*x*)

get_indexes(*xs*)

get_xx()

Listing conditional values

Provide a list of all sampled conditional values.

interpolate_x(*newxx*)

kind()

reset_cache()

scale_p(*a*)

Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

scale_y(*a*)

Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(rac{y}{a} | x)$.

set_x_cache_decimals(*decimals*)

write(*file*, *delimiter*)

write_file(*filename*, *delimiter*)

openest.models.model module

class openest.models.model.**Attribute**(*title*, *description*, *reference*, *subtitle*, *value*, *comments*, *source*)

Bases: object

An attribute is an arbitrary piece of information about a model, available from the attribute functions on Model.

class openest.models.model.**Model**(*scaled=True*)
Bases: object

Model class

Top level Model class, from which all specific model derive. All models should implement most of these functions (with the notable exceptions of merge and combine).

attribute_list()

static combine(*models*, *factors*)
Construct a weighted sum over the shared values of x

Each form provides methods for constructing the distribution of the sum of multiple parameters, which is generally constructed by performing the convolution: $p(y + z | x) = p_y(y | x) * p_z(z | x)$.

combiners = {'bin_model+bin_model': <function combine at 0x7fc306a65410>, 'bin_model+

copy()

draw_sample(*x=None*)
Produce a sample value of y from the conditional distribution.

eval_pval(*x*, *p*, *threshold=0.001*)
Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

get_attribute(*title*)

get_mean(*x=None*)
E[Y | X]

get_sdev(*x=None*)
sqrt Var[Y | X]

kind()

static merge(*models*)
Pooling Merging

Each form provides methods for producing a pooled parameter estimate from multiple parameter estimates. These could all be parameter estimates with the same form, or with two different forms: $p_1(y | x) p_2(y | x)$.

mergers = {'bin_model': <function merge at 0x7fc306a65398>, 'bin_model+ddp_model': <

scale_p(*a*)
Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

scale_y(*a*)
Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(

rac{y}{a} | x)$.

to_points_at(*x*, *ys*)
Conditional Probability Density Evaluation

Returns unscaled probability density values for given values of x and y: $f(y | x)$.

openest.models.multivariate_model module

class openest.models.multivariate_model.**MultivariateModel**(*xx_is_categoricals*, *scaled=True*)

Bases: *openest.models.model.Model*

condition(*conditions*)

default_condition()

numvars()

openest.models.outer_multi_model module

class openest.models.outer_multi_model.**OuterMultiModel**(*xxs*, *xx_is_categoricals*, *union*, *scaled=True*)

Bases: *openest.models.multivariate_model.MultivariateModel*

condition(*conditions*)

default_condition()

dims()

float_condition(*conditions*)

init_from_union(*union*)

kind()

static re_condition(*condition*)

re_numeric = <_sre.SRE_Pattern object>

scale_p(*a*)

Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

write(*file*, *delimiter*)

write_file(*filename*, *delimiter*)

openest.models.parameter module

class openest.models.parameter.**ParameterBase**(*title*, *units*)

Bases: object

derive(*subtitle*)

openest.models.spline_model module

class openest.models.spline_model.**SplineModel**(*xx_is_categorical=False*, *xx=None*, *conditionals=None*, *scaled=True*)

Bases: *openest.models.univariate_model.UnivariateModel*, *openest.models.memoizable.MemoizableUnivariate*

Model Spline File

Each line in a model spline file represents a polynomial segment in log-probability space. The format is as follows:

```
spp1
<x>,<y0>,<y1>,<a0>,<a1>,<a2>
...
```

Each line describes a segment of a probability distribution of y, conditional on x = <x>. The segment spans from <y0> to <y1>, where the lowest value of <y0> may be -inf, and the highest value of <y1> may be inf. The <x> values may also be categorical or numerical. If they are numerical, it is assumed that these values represent samples of a smoothly varying function (a cubic spline in every y).

The values <a0>, <a1> and <a2> are the polynomial coefficients in y (with quadratic coefficients, only normal or exponential tails are possible). The final segment of the probability function is:

```
exp(a0 + a1 y + a2 y2)
```

Parameters

- **xx_is_categorical** (*bool*) –
- **xx** (*list-like*) –
- **conditionals** –
- **scaled** (*bool*) –

add_conditional(*x*, *conditional*)

cdf(*xx*, *yy*)

static combine(*one*, *two*)

copy()

static create_gaussian(*xxs*, *order=None*, *xx_is_categorical=True*)
xxs should be a dictionary of the form {x: (mean, variance)}.

static create_single(*xxs*, *y0s*, *y1s*, *coeffss*, *order=None*, *xx_is_categorical=True*)

draw_sample(*x=None*)
Produce a sample value of y from the conditional distribution.

eval_pval(*x*, *p*, *threshold=0.001*)
Inverse CDF Evaluation

Returns the value of y that corresponds to a given p-value: $F^{-1}(p | x)$.

eval_pval_index(*ii*, *p*, *threshold=0.001*)

filter_x(*xx*)

static from_ddp(*ddp_model*, *limits*)

get_conditional(*x*)

get_mean(*x=None*)
E[Y | X]

get_sdev(*x=None*)
sqrt Var[Y | X]

get_xx()
Listing conditional values

Provide a list of all sampled conditional values.

init_from_spline_file(*file*, *delimiter*, *status_callback=None*)

interpolate_x(*newxx*)
Determines whether argument *newxx* a subset of index `xx`.

is_gaussian(*x=None*)

kind()

static merge(*models*)

neginf = -inf

posinf = inf

recategorize_x(*oldxx*, *newxx*)
Construct a new model with categorical x values 'newxx', using the conditionals currently assigned to categorical x values 'oldxx'.

samples = 1000

scale_p(*a*)
Raise the distribution to the power 'a' and rescales.

Returns modifies this model and returns it

Return type self

scale_y(*a*)
Rescaling of the Parameter Dimension

Produces a new conditional PDF with the y dimension scaled by a constant: $p(z | x) = p(rac{y}{a} | x)$.

to_ddp(*ys=None*)

to_points_at(*x*, *ys*)
Conditional Probability Density Evaluation

Returns unscaled probability density values for given values of x and y: $f(y | x)$.

write(*file*, *delimiter*)

write_file(*filename*, *delimiter*)

write_gaussian(*file*, *delimiter*)

write_gaussian_plus(*file*, *delimiter*)

class openest.models.spline_model.**SplineModelConditional**(*y0s=None*, *y1s=None*, *coeffs=None*)

add_segment(*y0*, *y1*, *coeffs*)

approximate_mean(*limits*)

static approximate_sum(*conditionals*)

static ascinv(*y*, *func*, *minx*, *maxx*, *threshold*)

cdf(*yy*)

convolve(*other*)

copy()

draw_sample()

evaluate(*ii*, *y*)

find_mode()

static **find_nearest**(*array*, *value*, *within*)

gaussian_mean(*ii*)

gaussian_sdev(*ii*)

get_pval(*p*, *threshold=0.001*)

is_gaussian()

static **make_conditional_from_spline**(*spline*, *limits*)

static **make_gaussian**(*y0*, *y1*, *mean*, *var*)

static **make_single**(*y0*, *y1*, *coeffs*)

nongaussian_x2px(*ii*)

nongaussian_xpx(*ii*)

partial_cdf(*ii*, *y1*)

static **propose_grid**(*conditionals*)

rescale()

rough_limits()

rough_span()

scale(*factor*)

scale_p(*a*)

scale_y(*a*)

segment_max(*jj*)

size()

to_points(*ys*)

openest.models.sum_multi_model module

class openest.models.sum_multi_model.**SumMultiModel**(*unis*)

Bases: *openest.models.multivariate_model.MultivariateModel*

openest.models.univariate_model module

class openest.models.univariate_model.**UnivariateModel**(*xx_is_categorical=False*, *xx=None*, *scaled=True*)

Bases: *openest.models.model.Model*

filter_x(*xx*)

get_xx()

Listing conditional values

Provide a list of all sampled conditional values.

interpolate_x(*xx*)

static intersect_get_model(*model*, *xx*)

static intersect_get_x(*xx_is_categorical*, *one_xx*, *two_xx*)

static intersect_x(*one*, *two*)

static intersect_x_all(*models*)

recategorize_x(*oldxx*, *newxx*)

openest.swapbin package

Submodules

openest.swapbin.swapmodel module

openest.swapbin.swapmodel.**find_bins**(*means*, *sdevs*, *beta*, *vcv*)

openest.swapbin.swapmodel.**swap_any**(*model*, *beta*, *vcv*, *dropbin*, *totals*)

openest.swapbin.swapmodel.**swap_bin**(*model*, *beta*, *vcv*, *dropbin*, *totals*)

openest.swapbin.swapmodel.**swap_spline**(*model*, *beta*, *vcv*, *dropbin*, *totals*)

openest.swapbin.swapmodel.**swap_values**(*means*, *sdevs*, *beta*, *vcv*, *dropbin*, *totals*)

openest.swapbin.transform module

openest.swapbin.transform.**swap_beta**(*beta*, *T*)

openest.swapbin.transform.**swap_vcv**(*V*, *T*)

openest.swapbin.transform.**transform**(*predcount*, *bins*, *dropbin*, *totals*)

Construct a transform matrix from an old set of predictors to a bin swapped set.

The intercept is assumed to be the first predictor.

Parameters

- **predcount** (*int*) – the number predictors, including the intercept.
- **bins** (*list[int]*) – the indices of the bins amongst the predictors.
- **dropbin** (*int*) – an index into *bins*
- **totals** (*float*) – the value that all bin values would sum to, e.g. *1* if the bins are indicators e.g., 365 if the bins are daily over a year

Module contents

CHAPTER 3

The Developers Guide

If you want to contribute to the project, this part of the documentation is for you.

3.1 Contributor's Guide

This document lays out guidelines and advice for contributing to this project. If you're thinking of contributing, please start by reading this document and getting a feel for how contributing to this project works. If you have any questions, feel free to reach out to either James Rising, Mike Delgado, or Justin Simcock.

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report by following the guidelines in this document: *Bug Reports*.
3. Write tests that demonstrate your bug or feature. Ensure that they fail.
4. Make your change.
5. Run the entire test suite again, confirming that all tests pass *including the ones you just added*.
6. Send a GitHub Pull Request to the main repository's `master` branch. GitHub Pull Requests are the expected method of code collaboration on this project.

3.1.1 Documentation Links

The documentation files live in the `openest_docs/` directory of the codebase. They're written in reStructuredText, and use Sphinx to generate the full suite of documentation.

When writing documentation, please do your best to follow the style of the documentation files.

Here is an reference example Python Module with resStructuredText in the docstring.

```python
"""Example NumPy style docstrings.

This module demonstrates documentation as specified by the `NumPy
Documentation HOWTO`_. Docstrings may extend over multiple lines. Sections
are created with a section header followed by an underline of equal length.

Example
-------
Examples can be given using either the ``Example`` or ``Examples``
sections. Sections support any reStructuredText formatting, including
literal blocks::

    $ python example_numpy.py


Section breaks are created with two blank lines. Section breaks are also
implicitly created anytime a new section starts. Section bodies *may* be
indented:

Notes
-----
    This is an example of an indented section. It's like any other section,
    but the body is indented to help it stand out from surrounding text.

If a section is indented, then a section break is created by
resuming unindented text.

Attributes
----------
module_level_variable1 : int
    Module level variables may be documented in either the ``Attributes``
    section of the module docstring, or in an inline docstring immediately
    following the variable.

    Either form is acceptable, but the two should not be mixed. Choose
    one convention to document module level variables and be consistent
    with it.


.. _NumPy Documentation HOWTO:
   https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

"""

module_level_variable1 = 12345

module_level_variable2 = 98765
"""int: Module level variable documented inline.

The docstring may span multiple lines. The type may optionally be specified
on the first line, separated by a colon.
"""


def function_with_types_in_docstring(param1, param2):
    """Example function with types documented in the docstring.
```

(continues on next page)

(continued from previous page)

```
    `PEP 484`_ type annotations are supported. If attribute, parameter, and
    return types are annotated according to `PEP 484`_, they do not need to be
    included in the docstring:

    Parameters
    ----------
    param1 : int
        The first parameter.
    param2 : str
        The second parameter.

    Returns
    -------
    bool
        True if successful, False otherwise.

    .. _PEP 484:
        https://www.python.org/dev/peps/pep-0484/

    """


def function_with_pep484_type_annotations(param1: int, param2: str) -> bool:
    """Example function with PEP 484 type annotations.

    The return type must be duplicated in the docstring to comply
    with the NumPy docstring style.

    Parameters
    ----------
    param1
        The first parameter.
    param2
        The second parameter.

    Returns
    -------
    bool
        True if successful, False otherwise.

    """


def module_level_function(param1, param2=None, *args, **kwargs):
    """This is an example of a module level function.

    Function parameters should be documented in the ``Parameters`` section.
    The name of each parameter is required. The type and description of each
    parameter is optional, but should be included if not obvious.

    If \*args or \*\*kwargs are accepted,
    they should be listed as ``*args`` and ``**kwargs``.

    The format for a parameter is::

        name : type
            description
```

(continues on next page)

(continued from previous page)

```
            The description may span multiple lines. Following lines
            should be indented to match the first line of the description.
            The ": type" is optional.

            Multiple paragraphs are supported in parameter
            descriptions.

    Parameters
    ----------
    param1 : int
        The first parameter.
    param2 : :obj:`str`, optional
        The second parameter.
    *args
        Variable length argument list.
    **kwargs
        Arbitrary keyword arguments.

    Returns
    -------
    bool
        True if successful, False otherwise.

        The return type is not optional. The ``Returns`` section may span
        multiple lines and paragraphs. Following lines should be indented to
        match the first line of the description.

        The ``Returns`` section supports any reStructuredText formatting,
        including literal blocks::

            {
                'param1': param1,
                'param2': param2
            }

    Raises
    ------
    AttributeError
        The ``Raises`` section is a list of all exceptions
        that are relevant to the interface.
    ValueError
        If `param2` is equal to `param1`.

    """
    if param1 == param2:
        raise ValueError('param1 may not be equal to param2')
    return True


def example_generator(n):
    """Generators have a ``Yields`` section instead of a ``Returns`` section.

    Parameters
    ----------
    n : int
        The upper limit of the range to generate, from 0 to `n` - 1.
```

(continues on next page)

(continued from previous page)

```
    Yields
    ------
    int
        The next number in the range of 0 to `n` - 1.

    Examples
    --------
    Examples should be written in doctest format, and should illustrate how
    to use the function.

    >>> print([i for i in example_generator(4)])
    [0, 1, 2, 3]

    """
    for i in range(n):
        yield i


class ExampleError(Exception):
    """Exceptions are documented in the same way as classes.

    The __init__ method may be documented in either the class level
    docstring, or as a docstring on the __init__ method itself.

    Either form is acceptable, but the two should not be mixed. Choose one
    convention to document the __init__ method and be consistent with it.

    Note
    ----
    Do not include the `self` parameter in the ``Parameters`` section.

    Parameters
    ----------
    msg : str
        Human readable string describing the exception.
    code : :obj:`int`, optional
        Numeric error code.

    Attributes
    ----------
    msg : str
        Human readable string describing the exception.
    code : int
        Numeric error code.

    """

    def __init__(self, msg, code):
        self.msg = msg
        self.code = code


class ExampleClass(object):
    """The summary line for a class docstring should fit on one line.

    If the class has public attributes, they may be documented here
```

(continues on next page)

(continued from previous page)

```
    in an ``Attributes`` section and follow the same formatting as a
    function's ``Args`` section. Alternatively, attributes may be documented
    inline with the attribute's declaration (see __init__ method below).

    Properties created with the ``@property`` decorator should be documented
    in the property's getter method.

    Attributes
    ----------
    attr1 : str
        Description of `attr1`.
    attr2 : :obj:`int`, optional
        Description of `attr2`.

    """

    def __init__(self, param1, param2, param3):
        """Example of docstring on the __init__ method.

        The __init__ method may be documented in either the class level
        docstring, or as a docstring on the __init__ method itself.

        Either form is acceptable, but the two should not be mixed. Choose one
        convention to document the __init__ method and be consistent with it.

        Note
        ----
        Do not include the `self` parameter in the ``Parameters`` section.

        Parameters
        ----------
        param1 : str
            Description of `param1`.
        param2 : :obj:`list` of :obj:`str`
            Description of `param2`. Multiple
            lines are supported.
        param3 : :obj:`int`, optional
            Description of `param3`.

        """
        self.attr1 = param1
        self.attr2 = param2
        self.attr3 = param3  #: Doc comment *inline* with attribute

        #: list of str: Doc comment *before* attribute, with type specified
        self.attr4 = ["attr4"]

        self.attr5 = None
        """str: Docstring *after* attribute, with type specified."""

    @property
    def readonly_property(self):
        """str: Properties should be documented in their getter method."""
        return "readonly_property"

    @property
    def readwrite_property(self):
```

(continues on next page)

(continued from previous page)

```
        """:obj:`list` of :obj:`str`: Properties with both a getter and setter
        should only be documented in their getter method.

        If the setter method contains notable behavior, it should be
        mentioned here.
        """
        return ["readwrite_property"]

    @readwrite_property.setter
    def readwrite_property(self, value):
        value

    def example_method(self, param1, param2):
        """Class methods are similar to regular functions.

        Note
        ----
        Do not include the `self` parameter in the ``Parameters`` section.

        Parameters
        ----------
        param1
            The first parameter.
        param2
            The second parameter.

        Returns
        -------
        bool
            True if successful, False otherwise.

        """
        return True

    def __special__(self):
        """By default special members with docstrings are not included.

        Special members are any methods or attributes that start with and
        end with a double underscore. Any special member with a docstring
        will be included in the output, if
        ``napoleon_include_special_with_doc`` is set to True.

        This behavior can be enabled by changing the following setting in
        Sphinx's conf.py::

            napoleon_include_special_with_doc = True

        """
        pass

    def __special_without_docstring__(self):
        pass

    def _private(self):
        """By default private members are not included.

        Private members are any methods or attributes that start with an
```

(continues on next page)

(continued from previous page)

```
        underscore and are *not* special. By default they are not included
        in the output.

        This behavior can be changed such that private members *are* included
        by changing the following setting in Sphinx's conf.py::

            napoleon_include_private_with_doc = True

        """
        pass

    def _private_without_docstring(self):
        pass
```

3.1.2 Bug Reports

Bug reports are hugely important! Before you raise one, though, please check through the GitHub issues, **both open and closed**, to confirm that the bug hasn't been reported before. Duplicate bug reports are a huge drain on the time of other contributors, and should be avoided as much as possible.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z